Formal Verification of Data-Obliviousness in Hardware

MBMV’23 – March 24, 2023
Technische Fakultät, Universität Freiburg

Lucas Deutschmann, Johannes Müller, Mo Fadiheh, Dominik Stoffel, Wolfgang Kunz
Dept. of Electrical & Computer Engineering, RPTU Kaiserslautern-Landau
“Cambrian explosion” of confidential computing technologies

- Cryptographic methods, fully homomorphic encryption
- Trusted execution environments, secure enclaves
- Microarchitectural security defenses
- ...

All of these target confidentiality, for different threat models

They all fail unless data-oblivious computing is supported throughout all levels of the system stack
**Data-Oblivious Computing:**

- Runtime,
- resource usage and
- memory access patterns

of the program are independent of confidential data.
Example

```python
def check_key(user_input):
    if user_input != SECRET_KEY:
        raise Exception(“Wrong key provided!”)
```

**Problem:**
String comparison operator compares one byte at a time, stops as soon as a mismatch is found

- Not data-oblivious!
- Attacker can deduce the secret byte by byte, by measuring the runtime
“Constant-Time” Programming

- Writing programs such that their runtime and resource usage is independent of confidential information

Contributions from the SW community

- Open-source libraries [BearSSL, https://bearssl.org/]
- DSLs [Cauligi’17]
- Verification Tools [Almeida’18]

Can SW (alone) fix the problem?
“Opening Pandora’s Box” [Vicarte’21]
- 7 microarchitectural optimizations that undermine the constant-time paradigm

Threat is real!
- Data memory-dependent prefetchers recently found in Apple A14, M1 and M1 Max devices
- Security breach found [https://prefetchers.info/]

How can we restore the trust in HW?
Verification of “constant-time” SW [e.g., Cauligi’20]

Guarantees regarding HW/SW relationship
- Data-Oblivious ISA Extension [Yu’20]
- HW/SW Contracts [Guarneri’20]
- RISC-V Cryptography Extension [https://github.com/riscv/riscv-crypto]

Almost no formal verification for data-oblivious HW in RTL
- Clepsydra [Ardeshiricham’17]
- XENON [Gleissenthall’21]
Data-obliviousness of transient instruction execution (resilience against Spectre, Meltdown, ...)

Our previous work: Unique Program Execution Checking (UPEC) [DATE’19, DAC’20, DAC’21, TC’22]

Exhaustive and scalable approach ✔

HW primitives for data-oblivious operations:
Operation execution is independent of timing and resource usage

Open problem !

This work: UPEC for data-independent timing (UPEC-DIT) [DAC’22]
UPEC-DIT:
Formal approach for detecting data-dependent timing in RTL designs

- Qualifies a microarchitectural ISA implementation
  - Determines the data-oblivious ISA subset of a given CPU
  - Provides guarantees for “constant-time” programming

- Leverages state-of-the-art commercial property checking

- Scalable to realistic designs (~700k state bits)

Found violations of data-obliviousness in security-conscious designs
Interval Property Checking (IPC) on 2-safety computational model

Major extension of UPEC

Previously: Verifies confidentiality of data-at-rest

**UPEC-DIT**: Verifies confidentiality of data-in-transit

- Tolerates legal propagations of secret data
- Detects data-dependent variations of the control behavior

Create an **implicit** representation of the HW’s control behavior

- In terms of semi-automatically determined set of control signals
FUs can be treated as **black boxes**: control signals are given by I/O spec

- Separate control and data
- Generate 2-safety model
- Formulate property

**assume:**

at $t$: State_Equivalence();
prove:
during $t..t+k$: Control_Output_Eq();
<table>
<thead>
<tr>
<th>Design</th>
<th>Data-Ind. Timing</th>
<th>#States</th>
<th>Proof Time (s)</th>
<th>Max. Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasicRSA-T100</td>
<td>X</td>
<td>532</td>
<td>&lt; 1</td>
<td>589</td>
</tr>
<tr>
<td>SHA1</td>
<td>✔</td>
<td>911</td>
<td>&lt; 1</td>
<td>306</td>
</tr>
<tr>
<td>SHA256</td>
<td>✔</td>
<td>1103</td>
<td>&lt; 1</td>
<td>296</td>
</tr>
<tr>
<td>SHA512</td>
<td>✔</td>
<td>2162</td>
<td>&lt; 1</td>
<td>329</td>
</tr>
<tr>
<td>AES1</td>
<td>✔</td>
<td>2472</td>
<td>4</td>
<td>994</td>
</tr>
<tr>
<td>AES2</td>
<td>✔</td>
<td>554</td>
<td>&lt; 1</td>
<td>819</td>
</tr>
<tr>
<td>FWRISCV MDS-Unit</td>
<td>(!)</td>
<td>331</td>
<td>&lt; 1</td>
<td>596</td>
</tr>
<tr>
<td>ZipCPU Div-Unit</td>
<td>X</td>
<td>142</td>
<td>11</td>
<td>1347</td>
</tr>
<tr>
<td>CVA6 Div-Unit</td>
<td>X</td>
<td>209</td>
<td>&lt; 1</td>
<td>580</td>
</tr>
</tbody>
</table>
Distinguish between legal and illegal timing variabilities w.r.t. “constant-time” programming

- **ISA-visible** timing variations are legal (e.g., stalls due to dependencies between instructions)
- **ISA-invisible**, operand value-dependent timing variations are illegal

Global analysis of I/O behavior of HW is not tractable

- White-box approach necessary:
- Control flow must be represented in terms of *internal* control signals

Must consider instructions under *any* pipeline context
Solution:

- Security-critical timing behavior is determined by a small set of control signals

- Which control signals must be considered?
  - Iterative procedure
  - Starts with all state-holding signals
  - Refines property by analyzing propagation alerts

- 2-safety computational model with symbolic starting state, by construction, excludes legal timing variations, e.g., RAW hazards
**State_Equivalence()**:  
- Constrains state bits to be equal in the two instances  
- *Except* for the operand sources  
  - usually, the register file and forwarded operands

$t_{wb}$ denotes time point when instruction under verification (IUV) finishes execution

**assume:**
- at $t$: $State_Equivalence()$;  
- at $t$: $IUV\_in\_ID\_Stage(type)$;

**prove:**
- during $t..t_{wb}$: $Control\_Equality()$;

![Symbolic Memory Diagram](image-url)
UPEC-DIT for Processors

Constraint

UPEC-DIT Proof

No $(Z'_1 = Z'_2) \land (Y_1 = Y_2)$

Data-Dependancy

Yes

Caused by IUV?

No

Proof fails?

No

Data-Independence

Yes

Check Counterex.

No

Caused by data?(1)

Yes

Remove variable from proof(2)
<table>
<thead>
<tr>
<th></th>
<th>FWRISCV</th>
<th>IBEX</th>
<th>IBEX (+DIT)</th>
<th>SCARV</th>
<th>CVA6</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Type Arith.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>R-Type Arith.</td>
<td>✔/X</td>
<td>✔/✔</td>
<td>✔/✔</td>
<td>✔/✔</td>
<td>✔/✔</td>
</tr>
<tr>
<td>Multiplication</td>
<td>✔/✔</td>
<td>✔/X</td>
<td>✔/✔</td>
<td>✔/✔</td>
<td>✔/✔</td>
</tr>
<tr>
<td>Division</td>
<td>✔/✔</td>
<td>✔/X</td>
<td>✔/✔</td>
<td>✔/✔</td>
<td>X/X</td>
</tr>
<tr>
<td>Load</td>
<td>(!)</td>
<td>(!)</td>
<td>(!)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Store</td>
<td>(!)/✔</td>
<td>(!)/✔</td>
<td>(!)/✔</td>
<td>X/X</td>
<td>X/✔</td>
</tr>
<tr>
<td>Jump</td>
<td>(!)</td>
<td>✔</td>
<td>✔</td>
<td>(!)</td>
<td>X</td>
</tr>
<tr>
<td>Branch</td>
<td>X/X</td>
<td>X/X</td>
<td>✔/✔</td>
<td>X/X</td>
<td>X/X</td>
</tr>
<tr>
<td>#State Bits</td>
<td>3086</td>
<td>1019</td>
<td>1021</td>
<td>2334</td>
<td>682849</td>
</tr>
<tr>
<td>Average Time</td>
<td>3s</td>
<td>2min</td>
<td>4min</td>
<td>3min</td>
<td>1h 36min</td>
</tr>
<tr>
<td>Worst-Case Time</td>
<td>4s</td>
<td>5min</td>
<td>7min</td>
<td>8min</td>
<td>3h 7min</td>
</tr>
<tr>
<td>Max. Mem (GB)</td>
<td>1.7</td>
<td>4.5</td>
<td>4.3</td>
<td>2.1</td>
<td>11.9</td>
</tr>
</tbody>
</table>
Superscalar RISC-V processor with FP support, a deep 10-stage pipeline and out-of-order execution

Results:
- Proved data-obliviousness for I-type arithmetic, R-type arithmetic, multiplication (and FP arithmetic)
- Data-dependent timing in Int. Division, FP Division and FP Sqrt
- Properties take up to 20 hours

Current work: Develop an inductive property for better scalability
 UPEC-DIT detected several unknown violations of data-obliviousness

 Scalable and largely automated for RTL designs
   Instruction-level granularity: Well-defined interface with security guarantees for the entire system stack
   Current work: Extend to and inductive property to ensure scalability even for complex systems

 Closes important gap in making HW a root-of-trust for entire system stack
Thank you for your attention!

Many thanks to many collaborators!

Jörg Bormann, Anna Lena Duque-Antón, Wolfgang Ecker, Mohammad Rahmani Fadiheh, Jason Fung, Tobias Jauch, Wolfgang Kunz, Johannes Müller, Dino Mehmedagić, Subhasish Mitra, Sayak Ray, Philipp Schmitz, Stian Gerlach Sørensen, Dominik Stoffel, Alex Wezel

The reported research was partly supported by BMBF ZuSE (Scale4Edge), 16ME0122K-16ME0124, by DFG SPP Nano Security, KU 1051/11-1, and by Intel (Scalable Assurance).


Github: https://github.com/mofadiheh/upec-boom-verification-suite